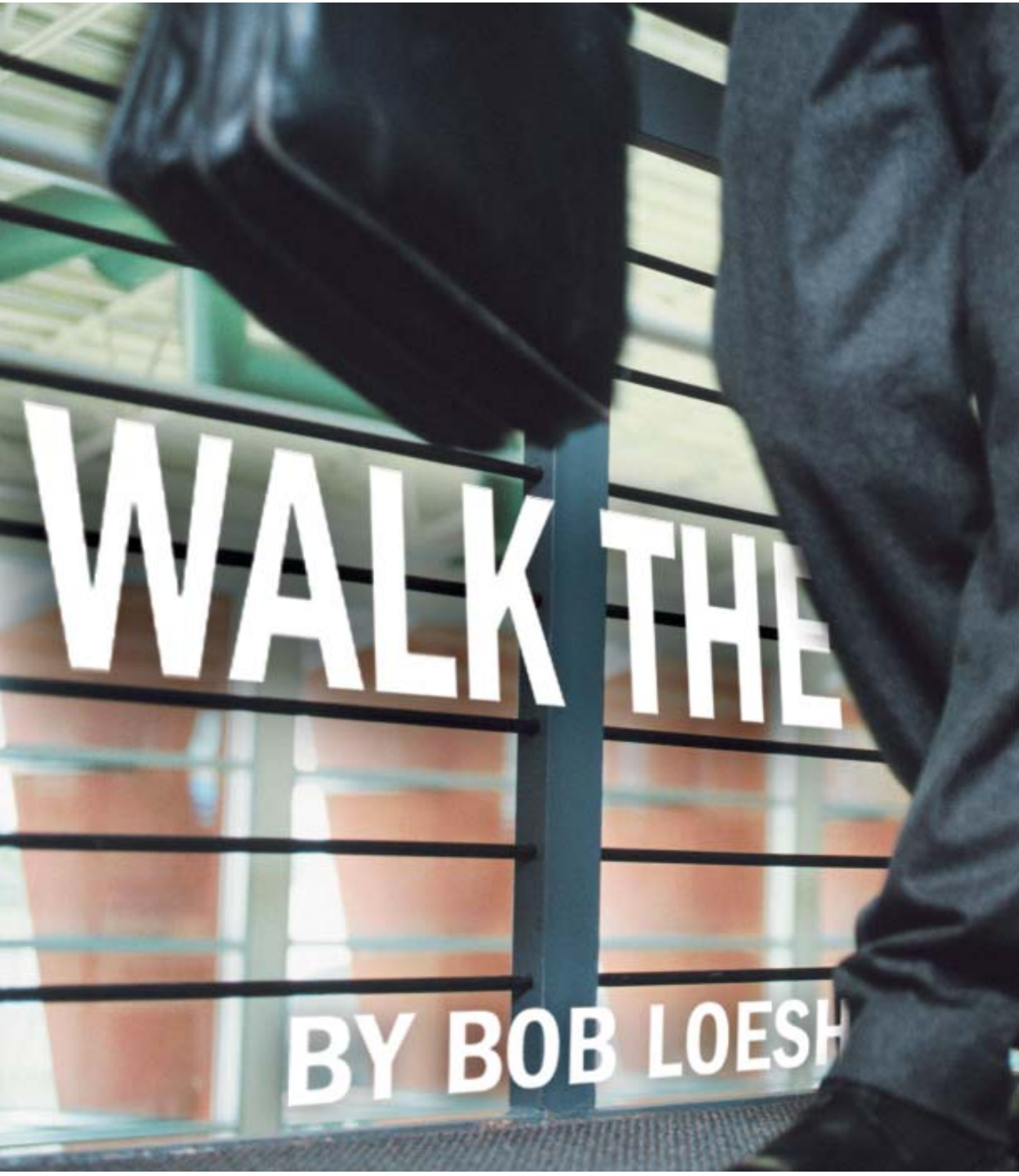
WALK THE
BY BOB LOESH


**I joined the Viking project in its early days in 1971 as the Orbiter Software System Engineer and stayed on board through several months after launch. Viking was a significant step in the technology of onboard computers and software for NASA and the Jet Propulsion Laboratory (JPL). But with those advances came many problems to solve. We've come a long way since then as well, but some of the fundamental issues, the problems we faced, and the solutions we found are all still relevant today.**

I WASN'T EXPECTING TO RETURN TO THE VIKING PROJECT, but as they were approaching the encounter, my division manager called me into his office and said, "Bob, I really need your help." When my division manager started talking to me like that, I knew I was going to say "yes" before I even heard what he needed me to do.

## PINPOINTING THE PROBLEM

The First-Order Image Processing and Enhancement System (VISRAP) group had run into some problems. The supervisor in charge of that group had gotten heavily involved in trying to get new state-of-the-art hardware to work and wasn't managing the software development too well. They were partially through coding and trying to integrate the software and were running into major difficulties. It just flat out didn't run.

One of the objectives was to get Viking Orbiter image data to the scientists as fast as possible so that they could make the best decision about where to land the Viking Lander on Mars. They wanted to make a pass of the planet in order to send back the Orbiter image data in real time. They would then process and enhance the images so they could use them to choose the best landing spot.

As soon as I came on board, I realized that the group's problems had less to do with technology than with poor development processes and communications. Due to the schedule pressure the programmers had stopped documenting design changes that they were putting into the code. I had run into this problem before on other projects. The programmers got so caught up in the coding and testing that the design documents were never updated, and the other team members had outdated versions of the interface design and functions between the design elements. When they went to test or use the software, new versions of the code had different functions and interfaces than the design documentation specified, and those that other team members were

**I CAME IN AND SLAMMED ON THE BRAKES. I SAID, "WE'RE TURNING THIS SHIP AROUND..."**

*The Viking lander model.*

using for their code. The design elements were incompatible and would not operate together.

By the time that I was called in, several months of not keeping the design documents updated had passed and the programmers couldn't remember all they'd designed and coded. They had to analyze the code to determine the correct interfaces and functions, and there were no updated design documents to help them. They also had not updated the test documents; so they had to spend five or ten times as long to fix a problem during integration that could've been easily solved if the documentation was current and correct. Months on the project had gone by like this. When the team put their software together for integration and testing: the software failed. There was no current documentation to help them understand why.

I saw it primarily as a management issue. The programmers and the other team members had not been given the direction, disciplined process, and motivation to ensure successful development and integration of the product as a whole. Staying on schedule was stressed as a major priority, and they lacked the focus to understand what it would take to deliver on time. They followed no system for coding and documentation; basically, there was no control.

## CRACKING THE CODE

So recognizing the problem is one thing, but solving it is another.

I understood how they had gotten themselves into this mess. In fact, it was learning from my own mistakes that helped me begin to tackle their problem. When I'm in the creative mode of coding, I can think of a million things I should have done better in the design phase. It takes a lot of discipline not to just throw the changes into the code without concurrently coordinating the design and updating the design and test documents. I knew I would have to do my best to provide the team with that same discipline.

The programmers were anxious about my takeover as soon as JPL management made the announcement. First, they didn't like me because I was an outsider. Second, these were programmers who previously had the freedom to do all the coding they wanted without documentation. Then I came in and slammed on the brakes. I said, "We're turning this ship around and going back to the drawing board."

The first thing I did was to shut everything down. I said, "There will be no more coding, designing, or fixing of errors until we've caught up the documentation." I'm still looking for the first programmer that would rather do documentation than code, so let's just say that they were not happy campers! I laid out a controlled process to keep these problems from repeating: coordinate the design to resolve the interface and incompatibility problems, document the agreed-upon design changes, and correct the code to reflect the coordinated and compatible design.

To do this, we had to keep programmers who knew—and hopefully could remember— what they had done huddled around the machine. It was a really inefficient way to do business, but my plan was for this to be the last time we'd be wading through all the old code. Now each time we found a problem, it was coordinated across the team and documented.

Because of this "catch up" process, several weeks were tacked onto our already stretched schedule to get the design understood, coordinated, and put to use. But I was convinced that we could make up some of the time by testing efficiency; we'd perform the team coordination and keep the design and test documents current. Each time they completed a certain amount of the design updates, we reviewed them together. They made it clear that they were still annoyed with me, but that was okay. We were on the road to recovery.

Once we had a handle on the documentation, they resumed coding and testing. I would schedule updates every two to three weeks to address changes that had been agreed upon by all affected staff. After the schedule was coordinated and everyone agreed to it, the entire team got a copy of our new plan.

When we got started on coding the coordinated changes, for a while I still went to their offices every day and asked them to "show me your documentation." My intent was not to micromanage, but to hammer home the importance of working as a team. They started doing it on their own, at first out of resentment to show me they could. But my strategy worked. Believe it or not, they began to see that

**I MANAGED THE PROJECT BY WALKING AROUND AND INTERACTING WITH THE TEAM.**

other people's documentation was useful. They could get things done quicker and with a lot less stress and effort.

After several weeks, the system started showing signs of working correctly during the integration testing. And for accomplishing the integration, the level of team efficiency improved by orders of magnitude. The success and pride that came from making the system work was a huge motivator.

## HERE TO STAY

At this point I continued to make it my job, several times a day, to hand-carry proposed changes to each person. I'd say, "Let's talk about these changes," and they'd tell me they didn't have the time. So I'd ask them, "What do you need to be able to get the time this afternoon?" Before long, people started realizing that I wasn't going away.

As I managed the project by walking around and interacting with the team, I got to know which people were a little quicker and which ones had more trouble. I also got to know which people weren't good at managing their workloads. I kept the lines of communication open about how much work each team member was carrying, and which person was the best choice to implement new changes.

I also got to actually see the work that was taking place rather than reading an email or hearing about it on the phone. It took a lot of personal time; but after making major process changes and overcoming huge setbacks, the last thing I wanted was for the project to fail because of bad work habits or the lack of interest on my part.

The time I invested paid off in the end. The images of Mars were delivered by this group to the scientists and mission designers on schedule, and they were used to accomplish a successful landing. •

### Lessons

- Discipline AND creativity are the keys to getting a software project completed on time.
- A direct contact, communication-by-walking-around management style can be the most effective control system.

### Question

*To be a good project manager, is it necessary that your team likes you?*

*An artist's rendering of the Viking spacecraft.*

### WATCH AND LEARN

**BOB LOESH has figured out a few things over the length of his 47-year career, which he began as a programmer at the RAND Corporation in 1957. Currently he is the Director of Engineering and Technology Development at Software Engineering Sciences, Inc., but the majority of his time was spent at the Jet Propulsion Laboratory working on high profile projects including Viking and Galileo. During that time, he served as NASA's "go-to" guy for software problems, uncovering what he believes to be the major obstacles keeping software project managers from reaching their full potential.**

**"Number one, we don't have any basic, formal training—either at the universities or in companies—for our software project manager people," says Loesh. "We put them on a project, they get along with people, they relate to management, and we promote them." But he says there is no training or formal way for them to learn techniques. And because of this, there are not many good models of successful software project managers for them to emulate.**

**This goes hand in hand with what Loesh describes as a second major problem: lack of mentorship. "We don't mentor our people. We don't pass along our experiences, guide them through problems, or let them watch what we do and learn from it," he says. "You look at occupations like bricklaying or machine work, and there's an apprenticeship. They do that for a couple of years and they learn all the right things to do."**

**Without formal training or a way to learn from of others, each manager is thrust into their software project with only the lessons of their own experience. Each new project manager continually recreates the wheel. Loesh adds, "We're going to repeat these problems over and over again unless we figure out a way to effectively train new software project managers."**